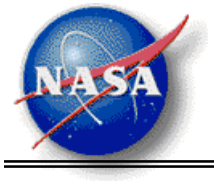


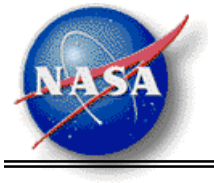
JPL

Failure Assessment

Robyn Lutz and Allen Nikora
November 9, 2005
ISHEM 2005

robyn.lutz/allen.p.nikora@jpl.nasa.gov
http://www.cs.iastate.edu/~rlutz

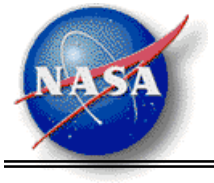
Some of the research described in this presentation was carried out at the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration, and was funded by NASA's Office of Safety and Mission Assurance. The first author's research is supported in part by National Science Foundation grants CCR-0204139 and CCR-0205588.



Failure Assessment

A *failure* is the inability of a system or component to perform its required functions within specified performance requirements [IEEE90].

***Failure assessment* is the identification and characterization of potential failure mechanisms in systems under development and of actual failure occurrences in operational systems.**



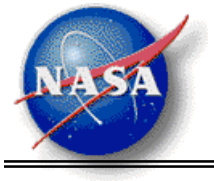
Topics in Failure Assessment

Three questions to which developers and users want accurate, precise answers are:

"How can the system fail?“

"What bad things will happen if the failure occurs?”

“How many failures will the system experience?”



SFMECA

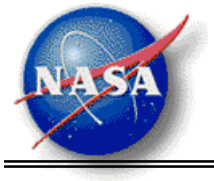
JPL

(Software Failure Modes, Effects & Criticality Analysis)

- Requirements or design analysis technique
- Structured *forward* analysis of the ways software can fail and the effects of these failure modes on the system and environment
- Originated with hardware FMEA (MIL-STD 1629A, 1984)
- Later adapted for software
- Use has grown as system reliability becomes more dependent on software
- Table-based & Guideword-driven ("timing wrong")

Example SFMECA

Data Item	*Failure Mode*	*Failure Description*	*System Effect*	*Criticality*
Heater ON	Timing wrong	Heater ON too early	Power allocation exceeded	Medium
Heater ON	Timing wrong	Heater ON too late	Experiment delayed	Low
Heater OFF	Timing wrong	Heater OFF too early	Science data lost	Low
Heater OFF	Timing wrong	Heater OFF too late	Power allocation exceeded	Medium

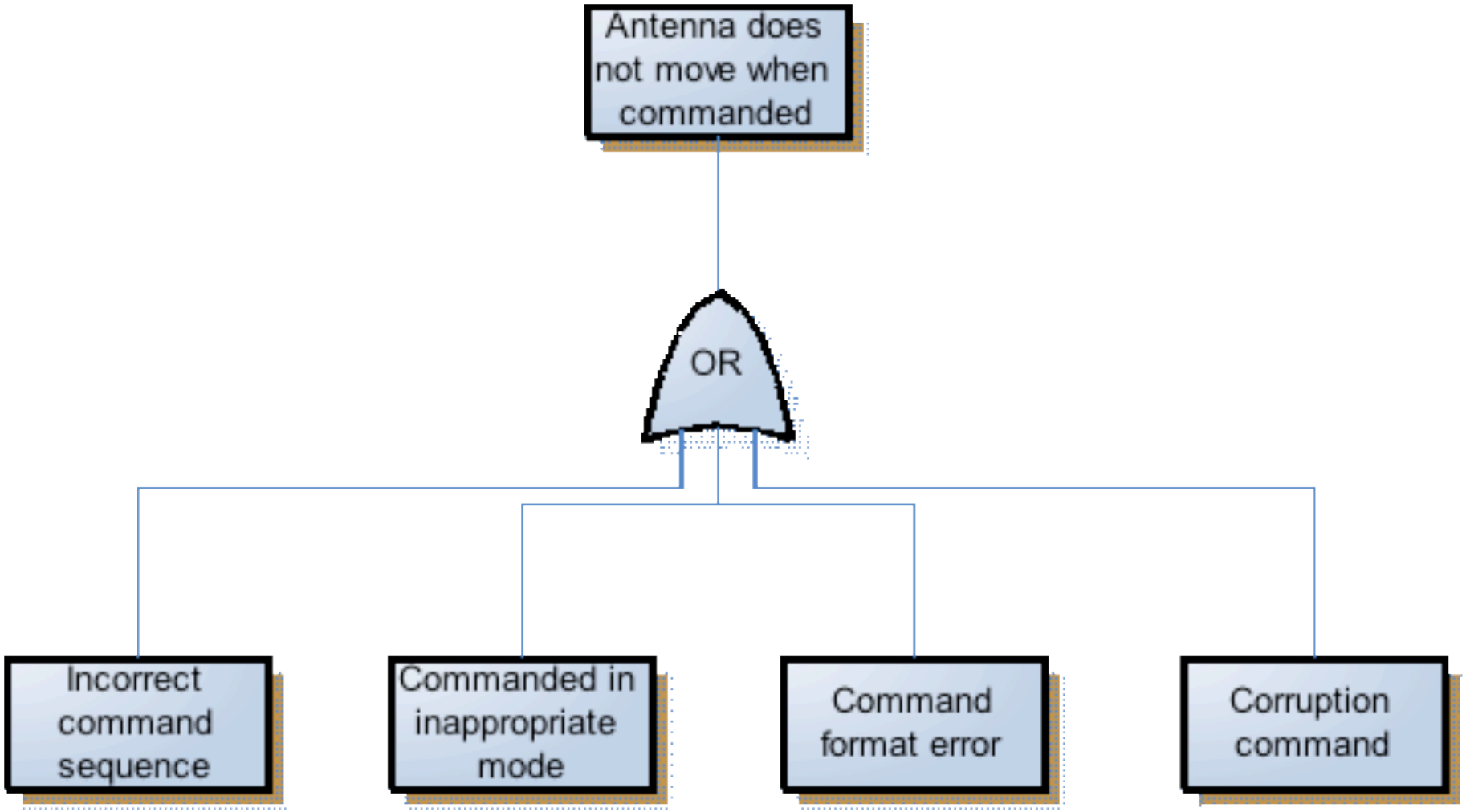


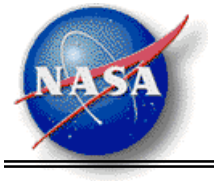
SFTA

(Software Fault Tree Analysis)

- Requirements, design or code analysis technique
- Structured ***backward*** analysis of the contributing causes to a root-node hazard or undesirable event
- Feasibility of occurrence investigated
- Originated with hardware; later adapted for software
- Use has grown as system reliability becomes more dependent on software
- Tree-based & uses Boolean logic (AND/OR gates)

Example SFTA





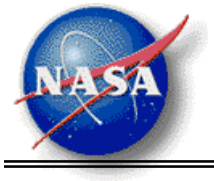
Why use SFMECA/SFTA?

JPL

- **JPL Software Development Process requires:**
  - Identification of possible failure modes
  - Analysis of possible failures of software and of interfacing software components,
  - Design consideration of off-nominal behavior
  - Removal of single-point failures
  - Prioritization of requirements for design attention
- **Cost recovered in failure and defect removal**
- **Broad choices in tool support**
- **Acceptance by system engineers**

Bi-directional Analysis

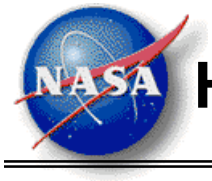
- ***Bi-directional Analysis*** **combines SFMECA with SFTA**
  - SFMECA: forward analysis identifies failure modes with unacceptable effects.
  - SFTA backward analysis evaluates feasibility and contributing causes of failure occurrence
  - Used on Galileo, Cassini, Autonomous Rotorcraft
  - NASA Software Safety Guidebook, NASA-GB-8719.13B notes that SFTA works well in conjunction with SFMEA
- **Example:**
  - Requirement: “Software shall try closing latch valves before firing pyro valves.”
  - Finding: Race condition can occur if overpressure occurs just prior to simultaneous enables at post-orbital insertion, resulting in pyro valve firing



Other Uses of Bi-directional Analysis

JPL

- **Product Lines**
  - Reuse of SFMECA/SFTA can be cost-effective for similar systems [DehlingerLutz04,06]:
  - Caveat: every system has its own peculiarities and operational environment
- **Security**
  - Attack trees are fault trees
  - Root node is “Intrusion”



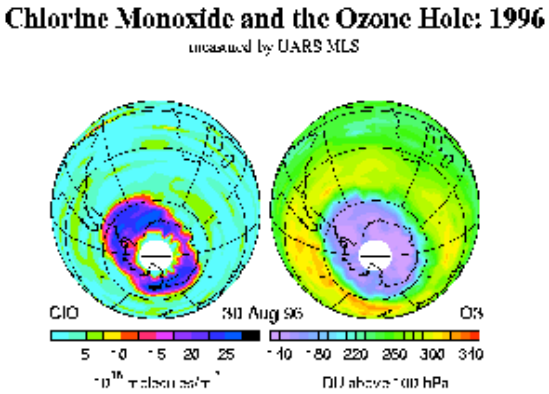
How many failures will the system experience?

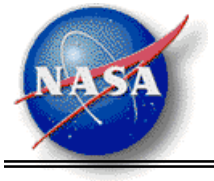
JPL

- **Software Reliability Engineering**
  - Software Reliability R(t): The probability of failure-free operation of a computer program for a specified time under a specified environment.
- **Software Reliability Modeling**
  - Focuses on design defects rather than physical wear-out
  - Estimates and forecasts the reliability of software systems
- Instrument code to assess risk of exposure to residual faults [Nikora03], e.g., as system evolves
- Estimate defect content from measures of software structure [Nikora04]

Experience on Mars Microprobe & EOS Microwave Limb Sounder

- Exploited existing analysis: leaf nodes involving software expanded into SFTAs
- Prioritized application: components responsible for fault detection, isolation, and response
- Integrated software/system analyses: looked for software contributors to component-level failures
- Useful findings (developers implemented fixes):
  - Sporadic hang-up
  - Missing information (e.g., timer value in anomalous case)
  - Loss of bus synchronization
  - Verified adequacy of handling of key faults

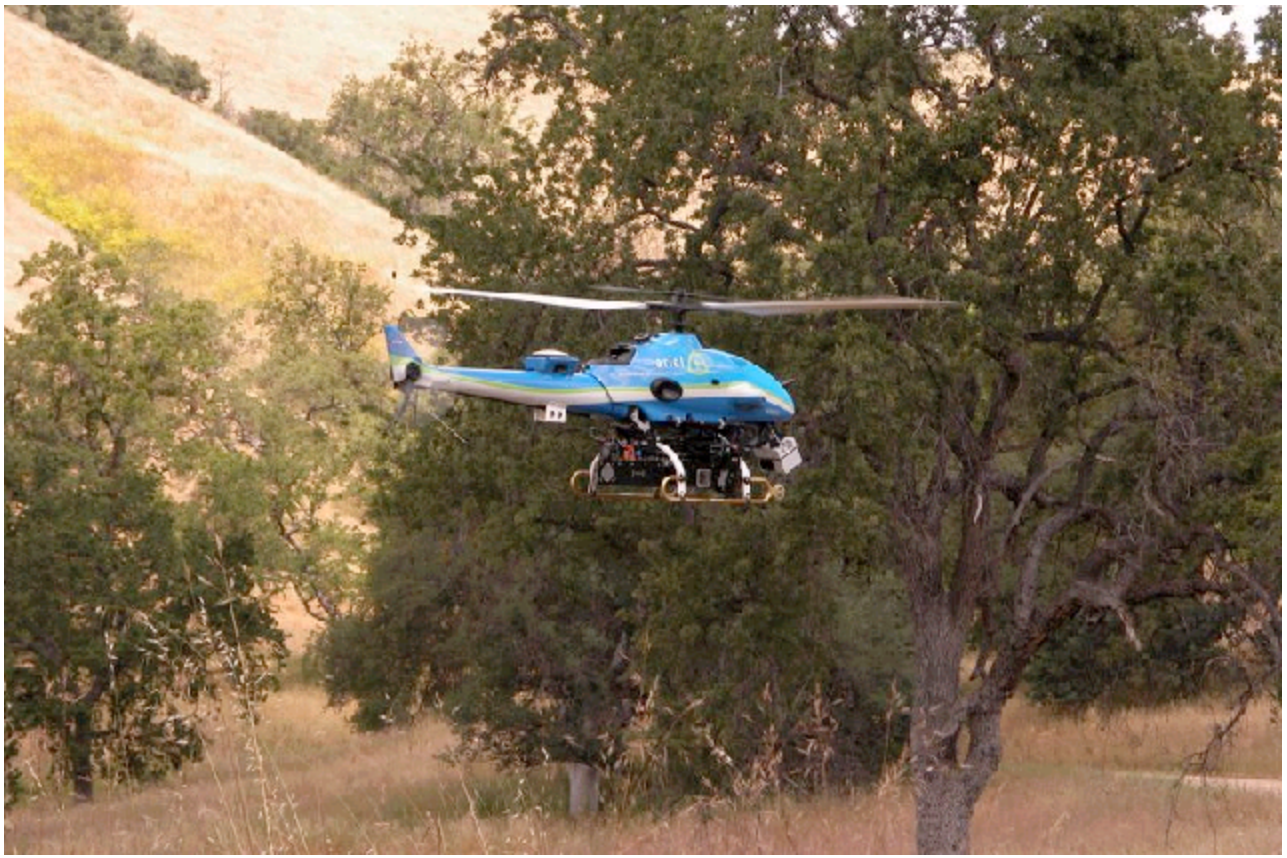




What Worked

- **Flexible use**
  - Use existing analyses as starting point
  - Zoom-in/zoom-out targeting
  - Dynamically choose next step
- **Risk-driven**
  - Project concerns are priority
  - Emphasize fault protection software
  - Fill gaps in analyses of embedded software
- **Integration of analyses**
  - Backward & forward analyses
  - System and embedded software analyses
- **Process adaptations**
  - Preserving traceability with tool support
  - Project interaction for quick changes

Using Bi-directional Analysis



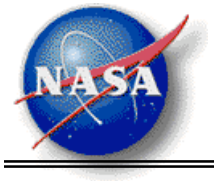
Autonomous Rotorcraft Project, NASA Ames
Matt Whalley, project lead
Collaborative work with Ann Patterson-Hine,
Rob Harris, Doron Tal, Anupa Bajwa, Stacy Nelson

Using Bi-directional Analysis

PROBLEM STATEMENT: Autonomous vehicles currently have a limited capacity to diagnose and mitigate failures. We need to be able to handle a broader range of contingencies.

A contingency is an event or condition (as an emergency) that may but is not certain to occur [Merriam-Webster]

- Example: Requirement to take camera images of landing site for autonomous landing.
- Contingencies
  - Failures: imaging of landing site fails due to hardware or software problem
  - Operational situations of concern: lens cap left causes all-black images
  - Environmental situations of concern: strong crosswind interferes with imaging



Using Bi-directional Analysis

JPL

1. Use SFTA, SFMECA, Obstacle Analysis to identify contingencies that risk mission-critical functions in Camera & Communication subsystems & identify mitigation or recovery actions;
2. Model contingencies & autonomous recovery actions using TEAMS (Testability And Engineering Maintenance System, QSI) toolset
3. Analyze contingencies: TEAMS produces diagnostic tree of checks needed to detect & isolate contingency, identifies missing checks and recovery actions
4. Code contingencies' diagnosis & recovery behavior in the rotorcraft's planner scripting language (auto-translation from TEAM's XML output)
5. Verify contingency scripts with hardware-in-loop simulation on the rotorcraft

Relevance to NASA

- Improved failure assessment and contingency handling needed to safely relinquish control of unpiloted vehicles to autonomous controllers
- Improved failure assessment and autonomous contingency handling needed to support extended mission operations